EE2703 : Applied Programming Lab
Assignment 4
Fourier Approximations

Sai Shashank GP
EE20B040

February 27, 2022

0.1 Introduction

In this assignment, we are going to analyse the Fourier series representation of two functions, namely, e^x and $cos(cos(x))$ and also try to approximate he two functions using Fourier series and Linear Algebra.
We are going to use *pylab* for plotting graphs, *numpy* for scientific calculations and array manipulations, *scipy* for definite integration and least sqaures method.

0.2 Assignment

0.2.1 Prerequisties

First, we need to import all the before mentioned modules as we like.

```
from pylab import *
import numpy as np
from scipy.integrate import quad
from scipy.linalg import lstsq
```

And we should also define some useful functions which might recur in the code

```
def exp(x):
    '''This function returns e^x'''
    return np.exp(x)

def ccx(x):
    '''This function returns cos(cos(x))'''
    return np.cos(np.cos(x))

def u(x, f, k):
    '''This function returns f(x)cos(kx) for given arguments'''
    return f(x)*np.cos(k*x)

def v(x, f, k):
    '''This function returns f(x)sin(kx) for given arguments'''
    return f(x)*np.sin(k*x)

def findCoeffs(f, n_iter):
    '''This function finds the fourier coefficients of a given function f(x) where n = n
    na = n_iter//2 + 1
    nb = n_iter//2
    f_a = np.zeros(na)
    f_b = np.zeros(nb)
    for i in range(1, na):
        f_a[i] = 1/np.pi * quad(u, 0, 2*np.pi, args=(f, i))[0]
    for i in range(nb):
        f_b[i] = 1/np.pi * quad(v, 0, 2*np.pi, args=(f, i+1))[0]
    f_a[0] = 1/(2*np.pi) * quad(u, 0, 2*np.pi, args=(f, 0))[0]
    return f_a, f_b
```

```
def constructCoeffVector(a_arr, b_arr):
    '''This function takes in a, b arrays and construct the actual coefficient sequence
    aList = list(a_arr)
    bList = list(b_arr)
    finalList = [aList[0]]
    aList.pop(0)
    i = 0
    for j in range(2*len(bList)):
        if j % 2 == 0:
            finalList.append(aList[i])
        else:
            finalList.append(bList[i])
            i += 1
    return np.array(finalList)

def splitCoeffVector(arr):
    '''This function takes in actual coeff vector and splits it into a, b vectors'''
    aList = [arr[0]]
    arrList = list(arr)
    arrList.pop(0)
    aList += arrList[::2]
    bList = arrList[1::2]
    return np.array(aList), np.array(bList)
```

0.2.2 Q1

Q1 wants us to plot the given actual functions

```
def Q1():
    '''This function executes Q1'''
    x = np.array(np.arange(-2*np.pi, 4*np.pi, 0.01))
    ccx_arr = ccx(x)
    exp_arr = exp(x)
    plot(x, ccx_arr)
    legend(['cos(cos(x))'], loc='upper right')
    grid(True)
    title('Plot of cos(cos(x))')
    show()
    semilogy(x, exp_arr)
    legend(['exp(x)'])
    grid(True)
    title('SemiLog Plot of exp(x)')
    show()
```

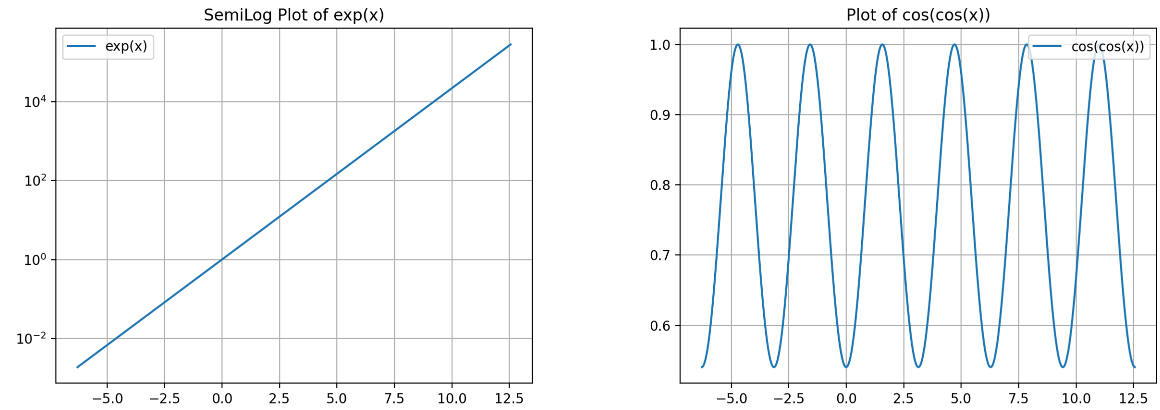

Figure 1: plots of two functions

0.2.3 Q2

Q2 wants us to calculate the first 51 a and b coefficients of those two functions.

```
n_iter = 51
global x1, x2
x1 = np.arange(0, n_iter//2 +1, 1)
x2 = np.arange(1, n_iter//2 +1, 1)

def Q2():
    ’’’This function executes Q2’’’
    global e_a, e_b, c_a, c_b
    e_a, e_b = findCoeffs(exp, n_iter)
    c_a, c_b = findCoeffs(ccx, n_iter)
```

0.2.4 Q3

Q3 wants us to plot the Fourier series coefficients of those two functions which we calculated before

```
 def Q3():
    ’’’This function executes Q3’’’
    Q2()
    figure(3)
    semilogy(x1, np.abs(e_a), ’ro’)
    semilogy(x2, np.abs(e_b), ’ro’)
    grid(True)
    xlabel(’n’)
    ylabel(r’$a_n, b_n$’)
    title(’SemiLog Plot of coeffs of exp(x)’)
    show()
    figure(4)
    loglog(x1, np.abs(e_a), ’ro’)
    loglog(x2, np.abs(e_b), ’ro’)
```

```
grid(True)
xlabel('n')
ylabel(r'$a_n, b_n$')
title('LogLog plot of coeffs of exp(x)')
show()
figure(5)
semilogy(x1, np.abs(c_a), 'ro')
semilogy(x2, np.abs(c_b), 'ro')
grid(True)
xlabel('n')
ylabel(r'$a_n, b_n$')
title('SemilLog Plot of coeffs of ccx(x)')
show()
figure(6)
loglog(x1, np.abs(c_a), 'ro')
loglog(x2, np.abs(c_b), 'ro')
grid(True)
xlabel('n')
ylabel(r'$a_n, b_n$')
title('LogLog Plot of coeffs of ccx(x)')
show()
```

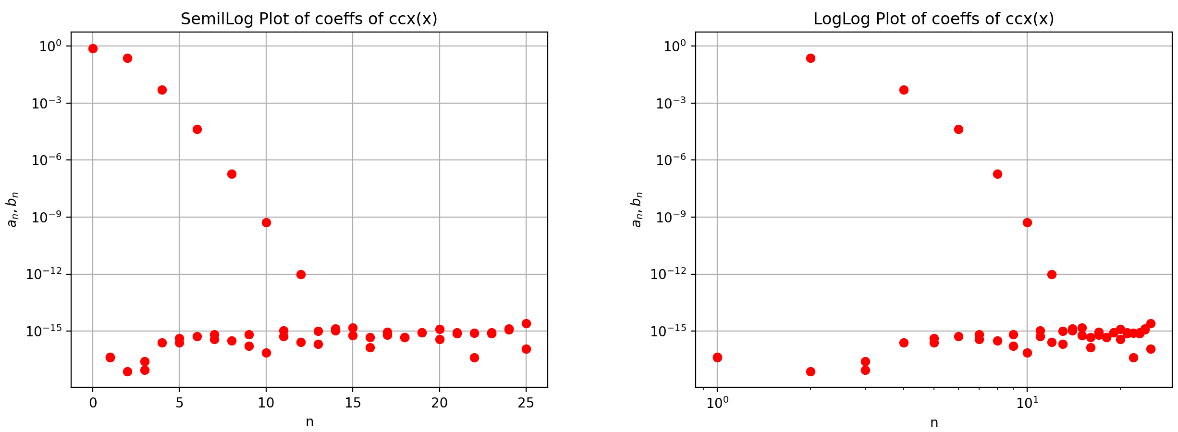

Figure 2: a_n, b_n semilog and loglog plot of $cos(cos(x))$

0.2.5 Answers for Q3

(a) The values of b_n coefficients are low because of the following property

$$b_n = \frac{1}{\pi}\int_0^{2\pi} cos(cos(x))sin(nx)dx b_n = \frac{1}{\pi}\int_0^{2\pi} cos(cos(x))sin(n(2\pi - x))dx \tag{1}$$

Therefore,

$$b_n = -b_n = 0 \tag{2}$$

(b) Let's try and find the values of a_n, b_n of e^x
Since we know the following

$$\int e^{ax} cos(bx) = \frac{e^{ax}}{a^2 + b^2}(bsin(bx) + acos(bx)) + c \tag{3}$$

$$\int e^{ax} sin(bx) = \frac{e^{ax}}{a^2 + b^2}(asin(bx) - bcos(bx)) + c \tag{4}$$

So, the following relation can be deduced for the coefficients of e^x

$$|a_n|, |b_n| \propto \frac{1}{n^2 + 1} \tag{5}$$

But for coefficients of $cos(cos(x))$, it is having quicker decay due to it's frequency. it is $\frac{1}{\pi}$. So, contribution of higher frequency cosines are negligible.

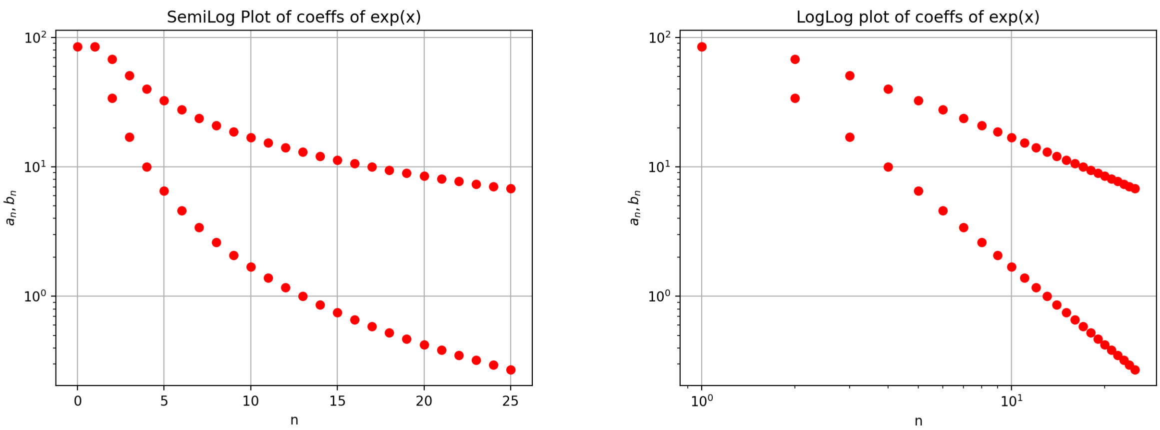

Figure 3: a_n, b_n semilog and loglog plot of e^x

(c) From (5), we can say that $|a_n|, |b_n|$ vary approximately linearly with $logn$
Since, the decay is quite fast in $cos(cos(x))$ case, it can be thought of as an exponentially decaying sequence and in log scale, it varies linearly with n.

0.2.6 Q4&5

Q4 wants us to calculate the coefficients without integration but my linear equation solving. Here, we take some 400 equidistant points in the interval $[0, 2\pi)$ and find out the actual values of those function at those points. Then we construct a vector with those values as elements. It's shape will be (400,). We need the coefficients which will also be a vector of shape (51,). We will construct a matrix of shape (400, 51) which contains the sine and cosine values of those 400 points with some modifications accordingly.
The final matrix equation is as follows

$$Ac = b \tag{6}$$

$$\begin{pmatrix} 1 & cosx_1 & sinx_1 & cos2x_1 & sin2x_1 & ... & cos25x_1 & sin25x_1 \\ 1 & cosx_2 & sinx_2 & cos2x_2 & sin2x_2 & ... & cos25x_2 & sin25x_2 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 1 & cosx_{400} & sinx_{400} & cos2x_{400} & sin2x_{400} & ... & cos25x_{400} & sin25x_{400} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ ... \\ a_{25} \\ b_{25} \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ ... \\ f(x_{400}) \end{pmatrix}$$

Solving this using least sqaures method, we get our best approximation of the coefficients of the function.

```
global x
x = np.linspace(0, 2*np.pi, 401)[:-1]
be = exp(x)
bc = ccx(x)
global A, ce, cc
A = np.zeros((400, n_iter))
A[:, 0] = 1
for k in range(1, n_iter//2 +1):
    A[:, 2*k-1] = np.cos(k*x)
    A[:, 2*k] = np.sin(k*x)
ce = lstsq(A, be)[0]
cc = lstsq(A, bc)[0]
cc_a, cc_b = splitCoeffVector(cc)
ce_a, ce_b = splitCoeffVector(ce)
def Q4():
    '''This function executes Q4&5'''
    Q2()
    semilogy(x1, e_a, 'ro', label='Actual coeffs')
    semilogy(x2, e_b, 'ro')
    semilogy(x1, ce_a, 'go', label='Calculated coeffs')
    semilogy(x2, ce_b, 'go')
    title('SemiLog Plot of actual & calculated coeffs of exp(x)')
    grid(True)
    legend()
    show()
    semilogy(x1, c_a, 'ro', label='Actual coeffs')
    semilogy(x2, c_b, 'ro')
    semilogy(x1, cc_a, 'go', label='Calculated coeffs')
    semilogy(x2, cc_b, 'go')
```

```
    title('SemiLog Plot of actual & calculated coeffs of ccx(x)')
    grid(True)
    legend()
    show()
```

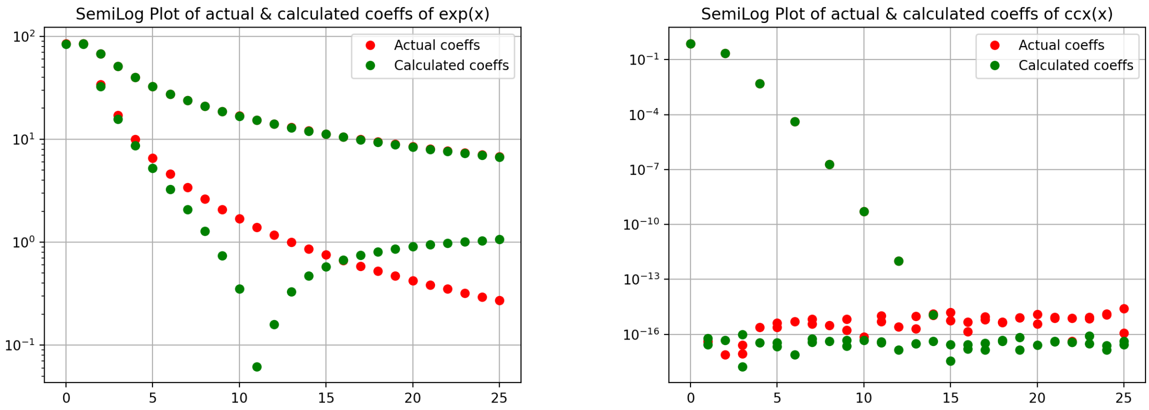

Figure 4: Actual vs Calculated coefficients of e^x and $cos(cos(x))$

0.2.7 Q6

Q6 wants us to compare the absolute deviation of the coefficients of those two functions.

```
def Q6():
    '''This function executes Q6'''
    ce_actual = constructCoeffVector(e_a, e_b)
    cc_actual = constructCoeffVector(c_a, c_b)
    maxdev_exp = np.max(np.abs(ce-ce_actual))
    maxdev_ccx = np.max(np.abs(cc-cc_actual))
    print(f'Max. deviation in coeffs of exp(x): {maxdev_exp}')
    print(f'Max. deviation in coeffs of ccx(x): {maxdev_ccx}')
```

```
Max. deviation in coeffs of exp(x): 1.332730870335368
Max. deviation in coeffs of ccx(x): 2.5461968575466718e-15
```

Figure 5: Max.absolute deviations in coefficients of each function

0.2.8 Answers for Q6

The maximum deviation in both the cases are quite small. In the case of $cos(cos(x))$, it was very small (of the order of 10^{-15})
Maximum deviation in coefficients of e^x case ~ 1.33
Maximum deviation in coefficients of $cos(cos(x))$ case $\sim 10^{-15}$

0.2.9 Q7

Q7 wants us to plot the actual functions along with the function which is obtained from the calculated fourier coefficients. Latter can be found out by multiplying the matrix A with obtained coefficients vector c.

```
def Q7():
    '''This function executes Q7'''
    calc_exp = np.matmul(A, ce)
    calc_ccx = np.matmul(A, cc)
    semilogy(x, exp(x), 'ro', label='Actual Function')
    semilogy(x, calc_exp, 'go', label='Calulated Function')
    legend(loc='lower right')
    grid(True)
    title('Actual vs Calculated of exp(x)')
    show()
    plot(x, ccx(x), 'ro', label='Actual Function')
    plot(x, calc_ccx, 'go', label='Calulated Function')
    grid(True)
    legend(loc='lower right')
    title('Actual vs Calculated of ccx(x)')
    show()
```

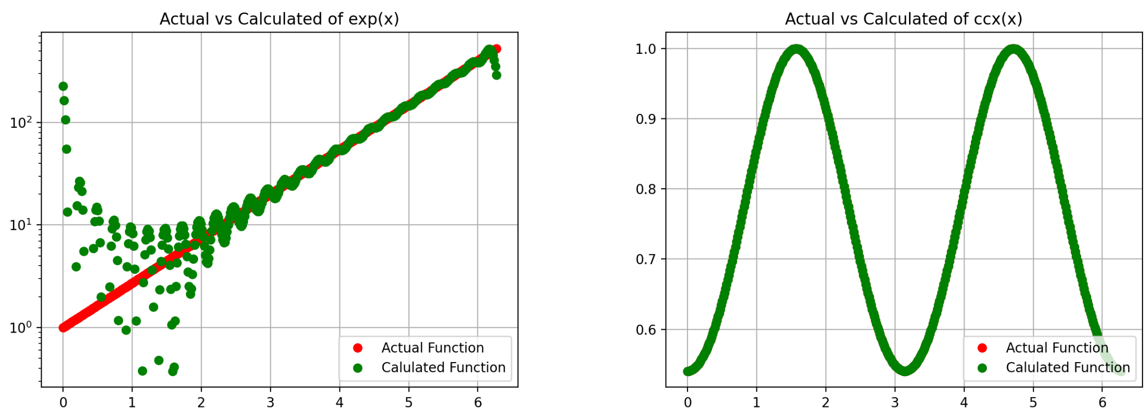

Figure 6: Actual vs Calculated functions

0.3 Conclusion

- When we try to approximate cosine functions with Fourier series, we get odd coefficients to be zero
- When we try to approximate non-periodic functions with Fourier series, we get the approximations to be scattered or in some random distribution at the point of discontinuities.

- When we take considerable amount of samples to approximately calculate Fourier coefficients, we get a good estimate of actual function but with above mentioned point applied
- Deviation in calculated coefficients is larger in the case of non-periodic signals compared to periodic signals

0.4 Takeaways

- I learnt how to seperate and combine the coefficient vector into a, b vectors
- I now understood about the Fourier approximations of non-periodic functions and the deviations is known as **Gibbs Phenomenon**
- I now got to know the easier way to approximate functions by using matrix properties and least squares method